加微信：642945106 发送“ 赠送 ”领取赠送精品课程
发数字“ 2 ”获取众筹列表

04 | 性能工程三定律：IT业和性能优化工作的“法律法规”

2019-12-04 庄振运

性能工程高手课 进入课程 >

讲述：庄振运
时长 14:33 大小 13.33M

你好，我是庄振运。

在开篇的几讲里，我谈了性能工程的重要性以及所需要的知识面，接下来我们就正式地进入相关的学习。

不过不要着急，第一个模块我们并不会直接进入性能问题的现场，一上来就去解决问题，而是要先耐下心来，学习一些必备的基础知识。为什么呢？因为学习任何事情，打好坚实的基础是至关重要的。

古人云：“合抱之木，生于毫末；九层之台，起于累土。”

所以接下来的几讲，我们需要先学习一些基础知识，包括和性能工程相关的几个重要定律法则和数理基础。这一讲我先和你探讨三个定律法则：帕累托法则、阿姆达尔定律和利特尔法则。

帕累托法则

我想你可能知道帕累托法则，它也被称为 80/20 法则、关键少数法则，或者八二法则。

这个法则是基于我们生活中的认识产生的，人们在生活中发现很多变量的分布是不均匀的——在很多场景下，大约 20% 的因素操控着 80% 的局面。也就是说，所有的变量中，比较重要的只有 20%，是所谓的“关键少数”。剩下的多数，却没有那么重要。

举例来讲，在企业销售中，根据帕累托法则，大约“80%的销售额来自 20%的客户”。认识到这一点对企业管理至关重要，比如需要重视大客户的关系。

虽然帕累托法则在生活中很多方面都适用，但我们今天的重点是来看看**帕累托法则是怎么应用到我们 IT 界**的，尤其是怎么指导我们的代码开发和性能优化相关的领域的。

我总结了一下，这个法则可以有以下几个领域的指导。

应用场景	帕累托法则内容	指导意义
应用程序的使用	80%的用户使用集中在大约20%的功能模块	对常用的模块进行充分优化
程序代码开发时间的分配	80%的开发时间往往花费在大约最精髓的20%代码	需要合理有序地规划好开发时间
程序代码的维护	80%的代码演进和改动发生在大约20%的代码上	需要对最常改动的少部分代码尽量熟悉
程序代码的修正和纠错	80%的错误发生在大约20%的代码上	代码修复和纠错时要重点修复最容易产生Bug的代码
客户流量的时间分布	80%的流量将在总时间段的特定20%内发生	应用程序设计和部署时，要充分考虑客户访问峰值和空闲时段
程序代码的优化	80%的时间是在运行大约20%的代码	发现少数重点代码并做优化

极客时间

1. 应用程序的使用

一个应用程序往往可以提供很多功能。但是，如果我们统计用户的使用情况，会经常发现大约 80% 的用户使用集中在 20% 的程序功能。

所以，对我们开发程序的指导意义是，要花足够心思在最常用的少数功能模块上，对它的设计和实现都要充分地优化。

2. 程序代码开发时间的分配

一个程序的开发过程中，大约 80% 的代码开发只占用了 20% 的总体开发时间。一般来讲，一个应用程序开发时，一开始往往花不了多少时间就可以快速地搭建一个大体可以工作的原型。反而是后面的剩余代码和代码改进需要更多的时间。

也就是说，80% 的开发时间往往用在最精髓的 20% 代码上。帕累托法则对我们的指导意义是，对代码开发工作量的安排上，需要合理有序地规划好开发时间。

3. 程序代码的维护

如果观察代码的维护和改进历史，你经常会发现少数代码被不断地改动，甚至经常被改得面目全非，而多数代码几乎从第一次写完后就一成不变了。按照帕累托法则，80% 的代码演进和改动发生在大约 20% 的代码上。

对程序维护而言，我们需要对这 20% 的代码尽量熟悉，这样才能对其他程序员的代码改动了如指掌，一目了然。

4. 程序代码的修正和纠错

统计数字也表明，所有的代码错误中，差不多有 80% 的错误发生在大约 20% 的代码上。

所以，根据帕累托法则，代码修复和纠错时要重点修复最容易产生 Bug 的代码，这样才能把保证整个应用程序的合理质量。

5. 客户流量的时间分布

防止断更 请务必加
首发微信：17716143665

估计 80%的流量将在总时间段的特定 20%内发生。比如一个商业软件，客户的流量峰值往往是上班时间开始的那几个小时（比如上午 9 点到 12 点）。

所以，我们在应用程序设计和部署时，要充分考虑帕累托法则带来的影响，尤其是客户访问的峰值时段和空闲时段。

6. 程序代码的优化

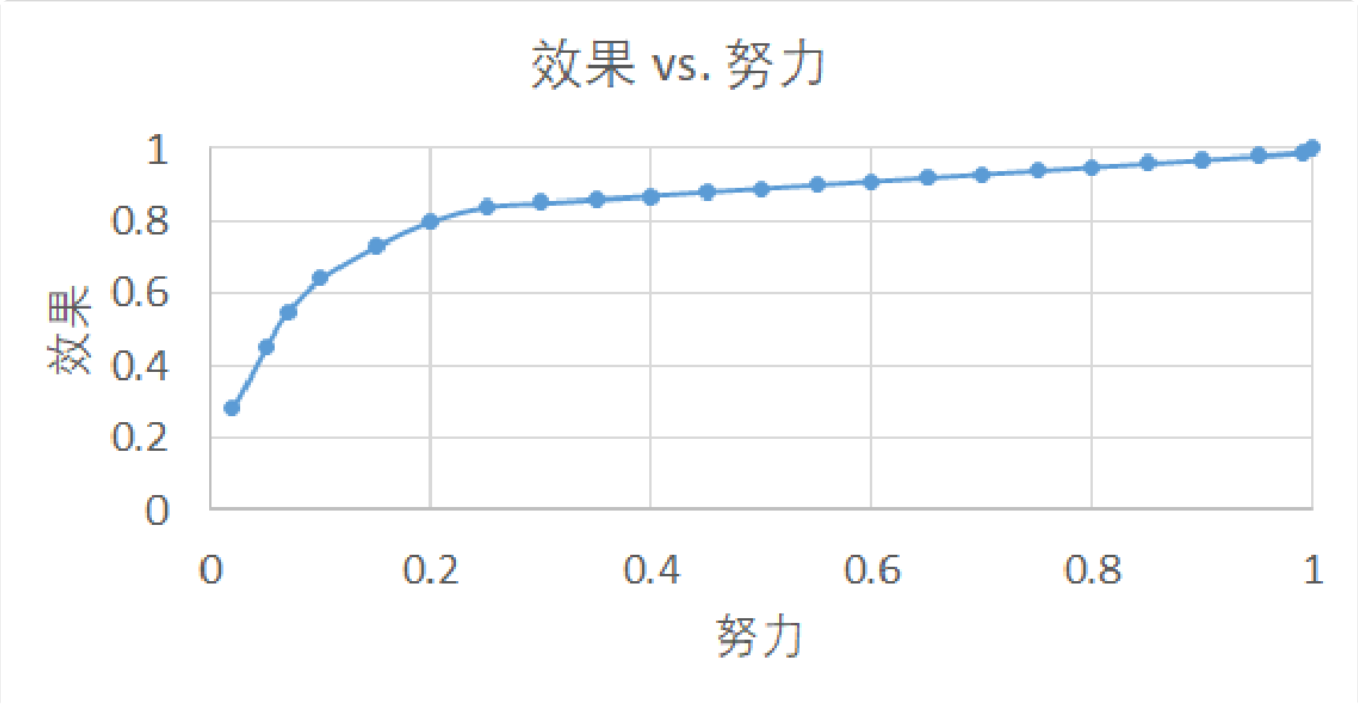
一个程序完成后必然会去运行，如果我们统计代码的运行时间，往往会发现程序的 80% 的时间是在运行大约 20% 的代码。也就是说，只有少数代码非常频繁地被调用。

所以，如果我们想提高程序的性能，最好找出这些少数代码，并做重点优化，这样就可以用很少的改动大幅度地提升整个程序和系统的性能。

那么现在，我们就从性能优化的角度，来看看如何参照帕累托法则，来规划我们性能优化工作的投入和产出。

假设我们的性能优化投入永远是按照代码的优先级来投入的，也就是说，总是要先优化最值得优化的代码。那么我们看到，只要投入差不多 20% 的努力，就能产出 80% 的性能优化产出，获得最大的投入产出比。

下图是一个根据帕累托法则来投入努力和产出效果的过程。



假如先解决 20% 的最重要的问题，就可以达到总体效果的 80%。以后再花费 80% 的努力，也只能解决剩下的 20% 的问题。

在应用帕累托法则的时候，需要注意的是，里面的 80% 或者 20% 都是大约数字，实际的场景千差万别，不可能是恰好这两个数字。这个法则的精髓是，我们的生活和自然界万物的分布不是均匀的，总有些因素比其他因素更重要。

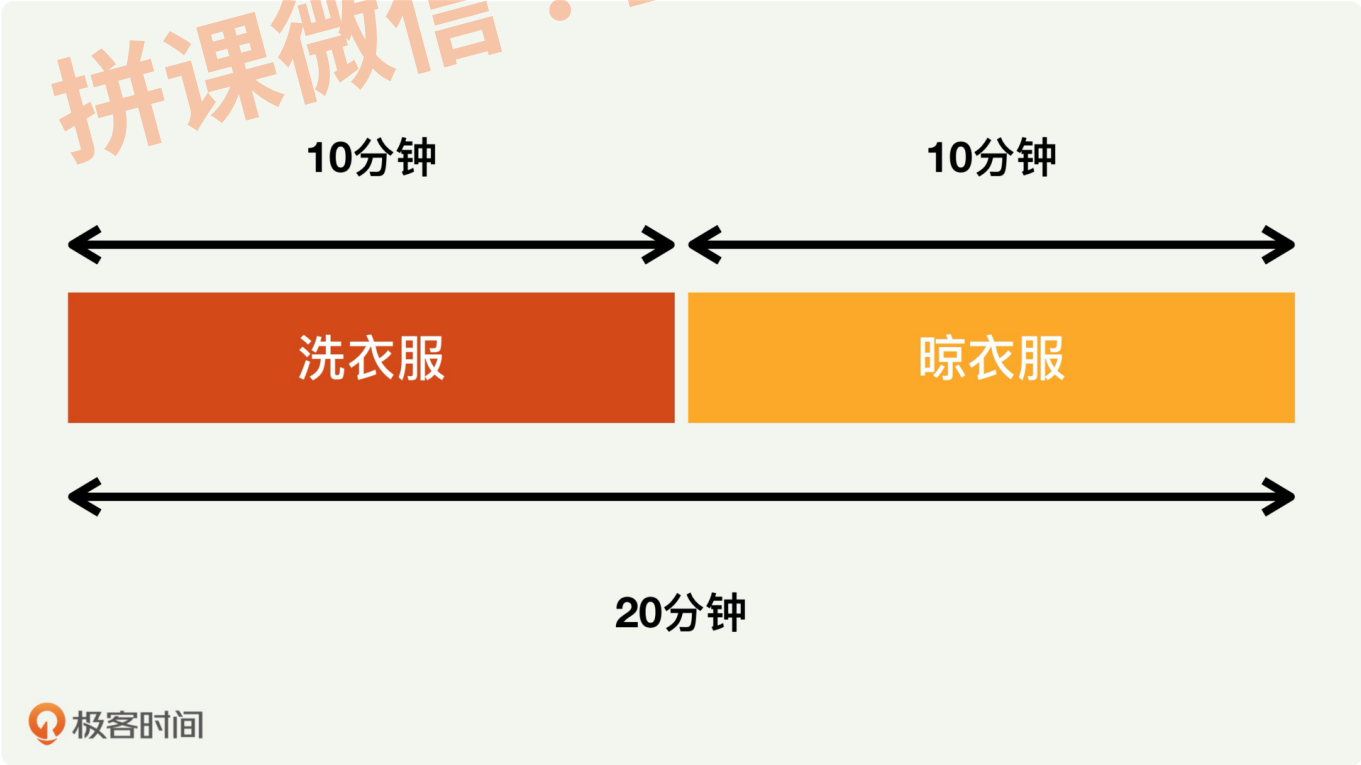
阿姆达尔定律

接下来我们来看第二个定律。这个定律你可能会感觉有点陌生。

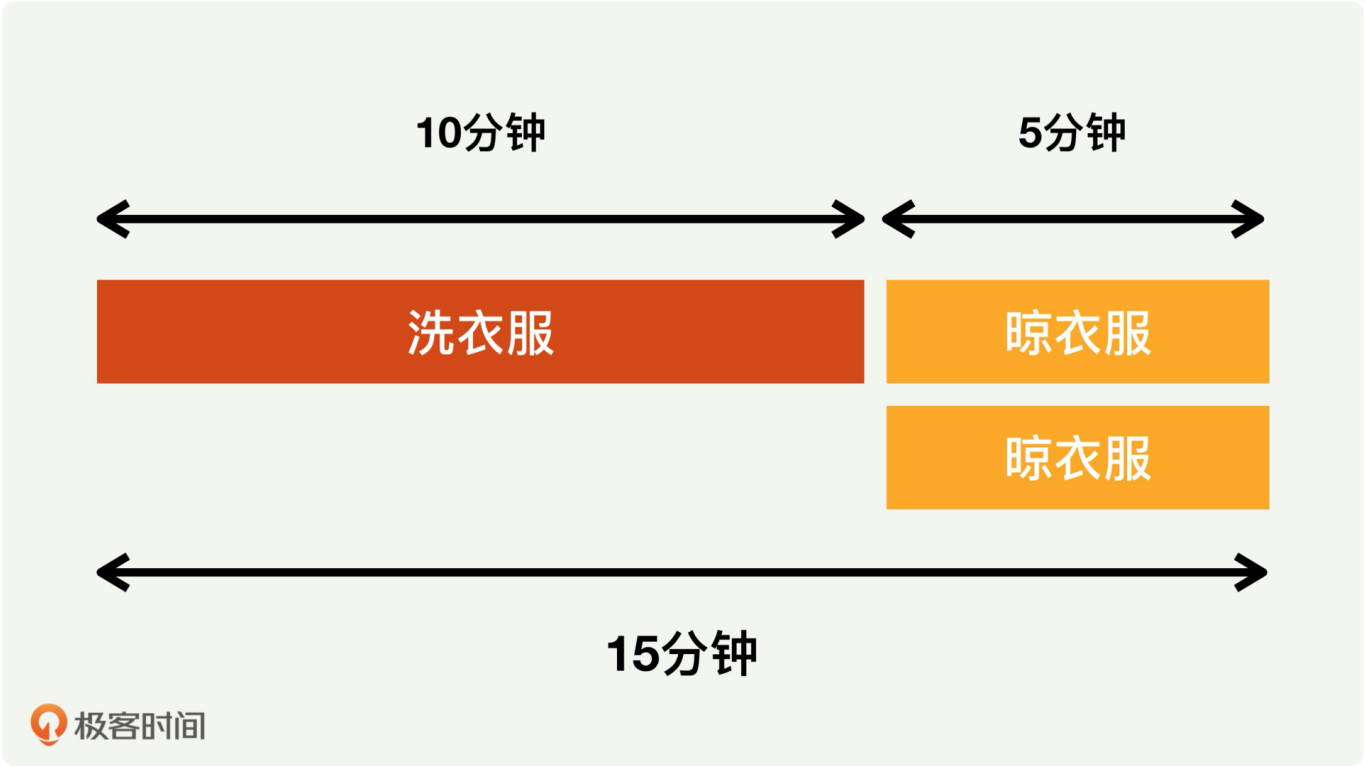
阿姆达尔定律（Amdahl’ s law / Amdahl’ s argument）是计算机科学界非常重要的一个定律和法则。它本来用于衡量处理器进行并行处理时总体性能的提升度。但其实阿姆达尔定律可以用在很多方面，为了方便你理解，我们就从一个简单的生活例子开始。

我们用洗衣服和晾衣服来举例。这里假设我们不用洗衣机，而是用传统的方式，先洗再晾。再假设洗衣服和晾衣服各需要 10 分钟，那么整个过程进行完需要 20 分钟。

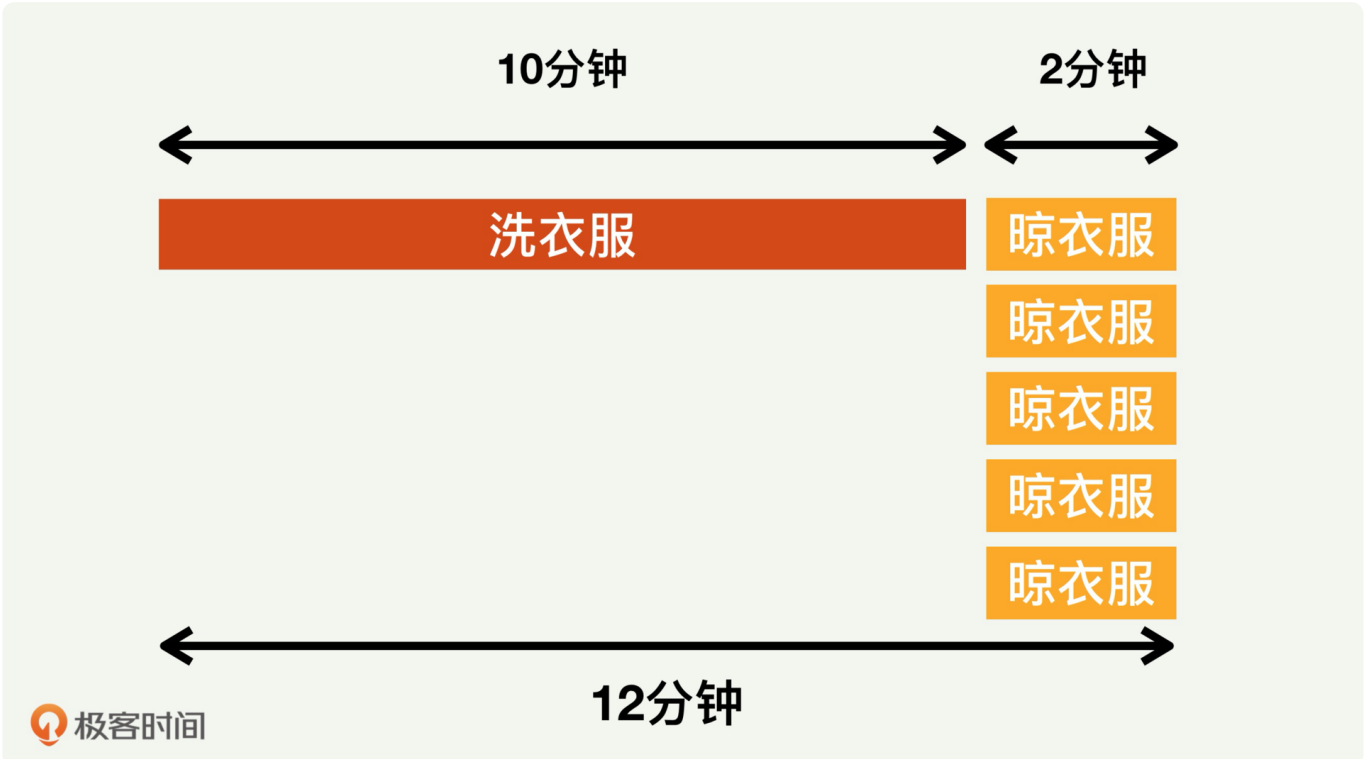
拼课微信：17161143665



如果我们对晾衣服的过程进行优化，从 10 分钟缩短到 5 分钟，相当于进行了两倍优化。现在整个过程需要多长时间呢？需要 15 分钟，因为洗衣服的模块还是需要 10 分钟。



在这个基础上，我们继续对晾衣服模块进行优化，速度提升 5 倍，从 10 分钟缩短到 2 分钟。整个过程现在需要 12 分钟完成。

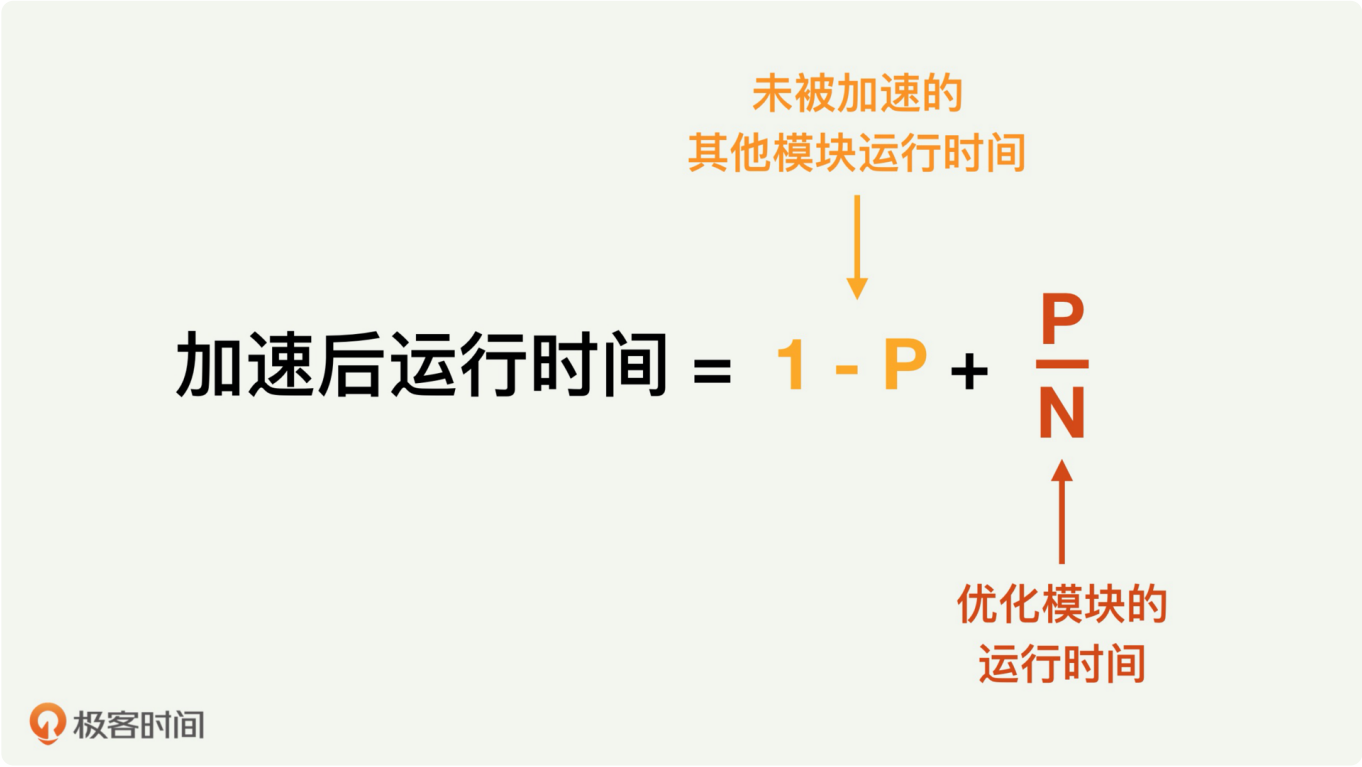


在这个基础上继续进行类推，我们就会发现，无论对晾衣服模块进行多大的优化，整个洗衣服、晾衣服的过程所需的时间不会小于 10 分钟，也就是整体加速比不会超过 2。

根据阿姆达尔定律描述，科学计算中用多处理器进行并行加速时，总体程序受限于程序所需的串行时间百分比。譬如说，一个程序 50% 是串行的，其他一半可以并行，那么，最大的加速比就是 2。无论用多少处理器并行，这个加速比不可能提高到大于 2。

所以在这种情况下，改进程序本身的串行算法可能比用多核处理器并行更有效。

用公式来讲，假设一个系统的整体运行时间是 1，其中要进行优化加速的模块运行用时是 P。如果对这个模块的加速比是 N，那么新系统的处理时间可以用下面的公式来表示。



这里面（1-P）是未被加速的其他模块运行时间，而 N 分之 P 是优化后的模块运行时间。它们的和就是新系统的总体运行时间。

相对于旧系统，运行时间的加速比就是：

$$加速比 = \frac{1}{加速后运行时间}$$

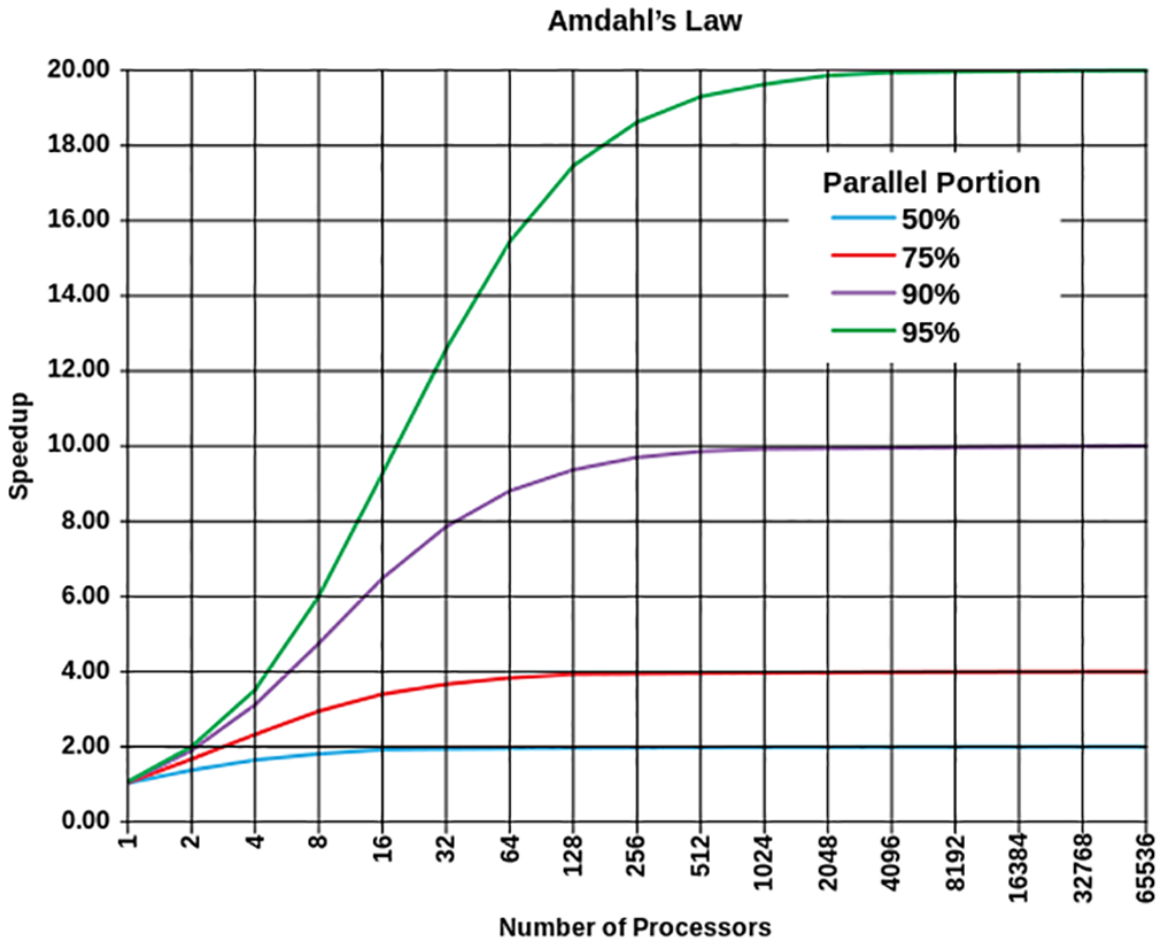
$$= \frac{1}{1 - P + \frac{P}{N}}$$

极客时间

阿姆达尔定律对我们进行性能优化的指导意义有以下 2 点。

1. 优先加速占用时间最多的模块，因为这样可以最大限度地提升加速比。
2. 对一个性能优化的计划可以做出准确的效果预估和整个系统的性能预测。

下面这张图描述了不同的并行百分比场景下分别进行并行优化的曲线。不同的曲线对应不同的并行模块百分比。横轴是并行程度，也就是多少个并行处理器。纵轴是速度提升度。



对每一条曲线我们都可以看到，超过一定的并行度后，就很难进行进一步的速度提升了。

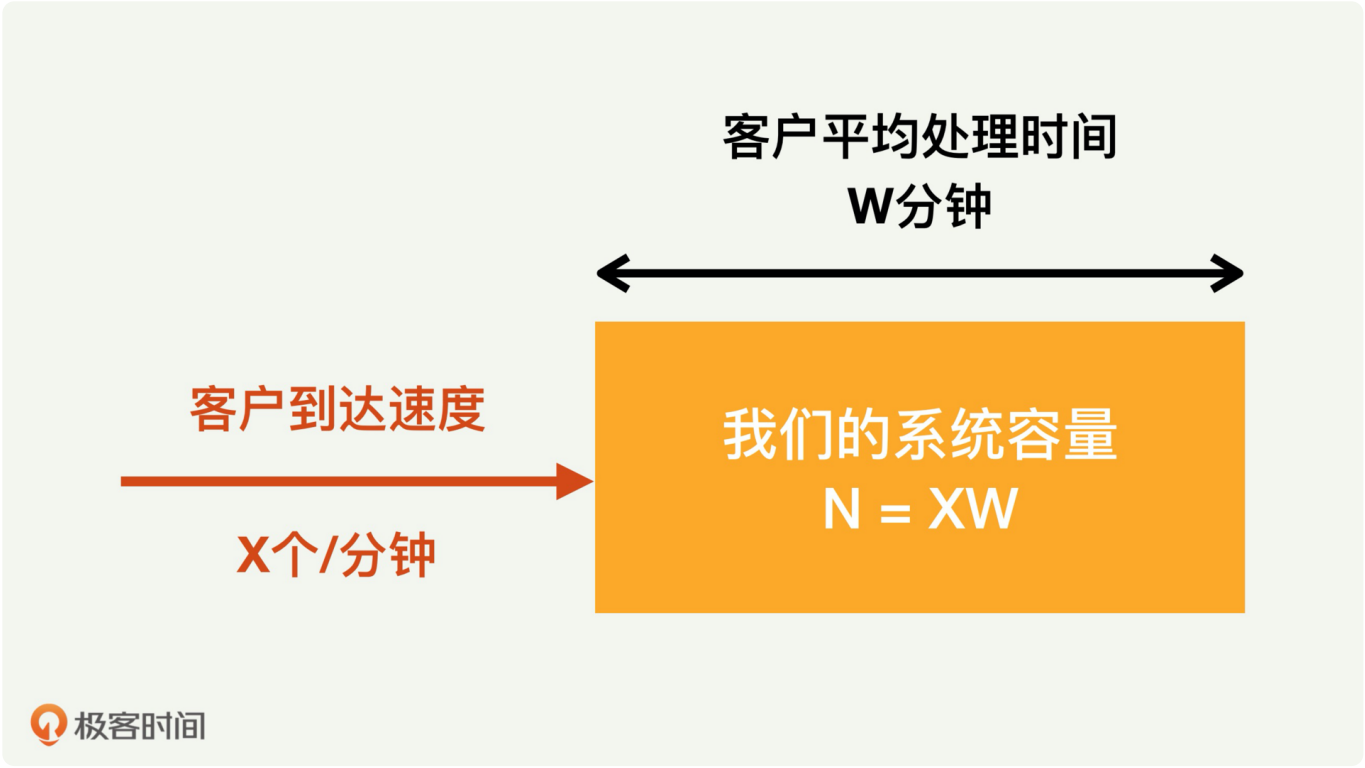
另外说明一点，阿姆达尔定律其实是另外一个定律的简化版本。这个更复杂的定律叫通用扩展定律（USL，Universal Scalability Law），你有兴趣的话可以去学习一下。

利特尔法则

接下来我们来看利特尔法则（Little’ s Law）。这个法则描述的是：在一个稳定的系统中，长期的平均客户人数（N）等于客户抵达速度（X）乘以客户在这个系统中平均处理时间（W），也就是说 N=XW。

这个法则看起来有点不直观，但从整个系统的宏观角度仔细想想的话就容易理解了。

如下图所示，客户按照一定的速度不断地进入我们的系统，假设这个速度是每分钟 X 个客户。每个客户在我们系统里的平均处理时间是 W 分钟。一旦处理完毕，客户就不会滞留在我们的系统里。



所以，如果这个状态稳定，也就是说，我们的系统处理速度恰恰好赶上客户到达速度的话，一方面系统没有空闲，另外一方面客户也不需要排队在系统外等待。那么在这个稳定状态下，我们的系统的总容量就恰好等于系统里面正在处理的客户数目。也就是说，N 就等于 X 和 W 的乘积。

我举一个服务器性能提升的例子来解释吧。

假定我们所开发的服务器程序可以进行并发处理，同时处理多个客户请求。并发的客户访问速度是**每分钟到来 1000 个客户**，每个客户在我们的服务器上花费的**平均时间为 2 分钟**。根据利特尔法则，在任何时刻，我们服务器系统里面将容纳 1000×2 = 2000 个客户。这 2000 个客户都在被服务。

过了一段时间，由于客户群的增大，并发的访问速度从每分钟 1000 客户增大到**每分钟 2000 个客户**。在这样的情况下，我们该如何改进我们系统的性能来应对呢？

根据利特尔法则，我们可以有两种方案来解决这一需求。

第一种方案是把客户的处理时间减半，从 2 分钟减到 1 分钟。这样我们的系统容量可以不变，客户滞留在我们系统的时间减半，刚刚好可以适应访问速率加倍的要求。系统容量就等于 2000 客户每分钟乘以 1 分钟，还是 2000 个客户。

第二种方案是扩大系统容量，维持处理时间不变。因为客户访问速度加倍了，所以系统容量也需要加倍，变成 4000。假如原来的系统需要 500 台服务器，那么新系统就需要 1000 台服务器。

从这里可以引申出利特尔法则在性能优化工作中的两种用处：

1. **帮助我们设计性能测试的环境。**性能测试的内容我们后面会详细讲到，这里简单提一下。比如当我们需要模拟一个固定容量的系统，那么性能测试的客户请求流量速度和每个请求的延时都需要仔细考虑。
2. **帮助我们验证测试结果的正确性。**有时候，如果性能测试的工作没有仔细地规划，得出的测试结果会出奇得好，或者出奇得差，从而让我们抓脑壳。这时如果采用利特尔法则，就可以很快地发现问题所在之处。

总结

我们今天讨论的性能工程相关的三大法则，分别是帕累托法则、阿姆达尔定律和利特尔法则。

可以说，这些法则就是 IT 业和性能优化工作的“法律法规”，有了它们，我们在实际工作中才能做到“有法可依，有法必依”。熟悉并熟练应用这几个法则，对我们的工作是会有很大的帮助的。

孟子说：“不以规矩，不能成方圆。”熟悉并熟练应用这几个“规律法则”，对我们的工作是会有很大的帮助的。

思考题

帕累托法则可以用到很多场景下，除了本讲中讨论的场景，你还能想到什么场景可以使用帕累托法则呢？使用这个法则会帮助你对问题的把握和找寻解决思路吗？

欢迎你在留言区分享自己的思考，与我和其他同学一起讨论，也欢迎你把文章分享给自己的朋友。

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇　03 | 导读：专栏是怎么设计的？需要哪些知识？

下一篇　05 | 概率统计和排队论：做性能工作必须懂的数理基础

精选留言 (6)

写留言

:)

2019-12-04

这三个法则的专业名称有点太难记住，，，利用老师给的例子，给三个法则定义别名，或许更容易记住

1. 20-80法则(帕累托法则)。告诉我们抓住重点，这就引申出另外一个问题，那20%如何定位?

2. 凉衣服法则(阿姆达尔定律)。告诉我们抓住总体的串行流程。...

展开

作者回复: 嗯，还真是可以考虑，毕竟外国人的名字太难记。

3

胡波 allenhu
2019-12-04

请问老师，系统容量和系统吞吐率的关系是什么?

展开

作者回复: 系统容量其实含义有点广，指的是一个系统最大处理能力。如果用请求数来衡量，意思是，一个系统（比如程序+服务器)的最大请求处理能力，比如每秒最多处理1000个请求。或者也可以说成是“系统最大吞吐率”。
有时候系统容量也可以用来衡量存储能力。比如一台服务器可以存1TB数据。
总之，“系统容量”往往指的是一个系统的性能瓶颈能力，可以是QPS（Query Per Second，也就是请求数），网络处理能力，存储能力等等。

2

Q
2019-12-05

1. 80-20法则可以指导我们，找到问题的关键，并形成“焦点”。
2. 阿尔达姆定律有点像短板理论，类似生活中的差等生成绩拖了全班平均成绩一样，优等生再努力，成绩都会有一个极限，推导到PC性能提升，机械硬盘往往是哪个短板。
3.利特尔法则关注的是一段时间系统能“同时处理”的请求数。

展开

1

张智凯
2019-12-06

linux的tcp接收缓存区大小 就是根据qps * 时延设的
80%的问题都是常见问题，20%的知识就可以解决 233

许童童
2019-12-05

老师讲的这几个法则非常好，帕累托法则、阿姆达尔定律和利特尔法则，记住这三个法则，就可以出去吹牛批，吊打小伙伴了。

展开

作者回复: ☺，学以致用，很好

行下一首歌

2019-12-04

嗯嗯，老师的这边专栏，高屋建瓴。我之间虽然知道用户访问量增大，可以通过横向扩容服务器来承受流量，但并没有理论上的认知。这下是学习到了。

展开

作者回复: 赞